January 12, 1999

WESTERLUND POWELL, P.C.
122 N. ALFRED STREET
ALEXANDRIA, VA 22314-3011
TEL. 703 706.5862/3
FAX 703.706.5860
e-mail:wp@us-patent-law.com

ASSISTANT COMMISSIONER FOR PATENTS
Box Patent Application
Washington, D.C. 20231

Dear Sir:

Transmitted herewith for filing under 35 U.S.C. 111(a) and 37 C.F.R. 1.53(b) is a new nonprovisional utility patent application for an invention entitled:

METHOD FOR INTERACTIVELY VIEWING FULL-SURROUND IMAGE DATA AND APPARATUS THEREFOR

and invented by: **Ford OXAAL**

The Attorney Docket No. is: **RP990001US**

If a CONTINUING APPLICATION, the appropriate box is checked below and the requisite information supplied:
☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application no.: _____
Prior Application Information: Examiner _____ Group/Art Unit _____

Enclosed are:

Application Elements

1. ☒ Filing fee as calculated and transmitted as described below.
2. ☒ Specification [Total pages 27], and including the following:
   -Descriptive Title of the Invention
   -Statement Regarding Federally-sponsored Research/Development
   -Field of the Invention
   -Background of the Invention
   -Summary of the Invention
   -Brief Description of the Drawings
   -Detailed Description
   -Appendix Containing Nine (9) Pages of Computer Code Listing
   -Claim(s) as Classified below
   -Abstract of the Disclosure
3. ☒ Drawings (35 U.S.C. 113) [Total Number of Sheets 13]
4. ☒ Oath or Declaration [Total pages 2]
   a. ☐ newly executed (original or copy) ☒ unexecuted
   b. ☐ copy from prior application (37 CFR 1.63(d))(for continuation/divisional application only)
      i. ☐ Deletion of Inventor(s)(Signed Statement attached deleting inventor(s) named in the prior application, see 37 CFR §§1.63(d)(2) and 1.33(b)).
5. ☐ Incorporation By Reference (usable if Box 4b is checked)
   The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference herein

Application Elements (Continued)

6. ☐ Computer program in Microfiche (Appendix)
7. ☐ Nucleotide and/or Amino Acid Sequence Submission (if applicable, all must be included)
   a. ☐ Paper Copy
   b. ☐ Computer Readable Copy (identical to computer copy)
   c. ☐ Statement Verifying Identical Paper and Computer Readable Copy

Accompanying Application Parts

8. ☐ Assignment papers (cover sheet and document(s))
9. ☐ 37 CFR 3.73(b) Statement (when there is an assignee) ☐ Power of Attorney
10. ☐ English Translation Document (if applicable)
11. ☐ Information Disclosure Statement (IDS)/PTO-1449 ☐ Copies of IDS Citations
12. ☐ Preliminary Amendment
13. ☒ Acknowledgement Postcard
14. ☒ Small Entity Statement(s) ☒ Statement filed in prior application, Status still proper and desired
15. ☐ Certified Copy of Priority Documents(s) (if foreign priority is claimed)
16. ☐ Additional Enclosures: _____

Fee Calculation and Transmittal

(SMALL ENTITY)

CLAIMS AS FILED					
For	#Filed	#Allowed	#Extra	Rate	Fee
Total Claims	19	20	0	$18.00	$0.00
Indep. Claims	3	3	0	$78.00	$0.00
Multiple Dependent Claims (check if applicable) ☐				x $260	$0.00
				BASIC FILING FEE	$380.00
OTHER FEE (specify purpose)_____					$ 0.00
				TOTAL FILING FEE	$380.00

☐ A check is attached in the amount of **$380.00** to cover the Total Filing Fee is enclosed.

☒ The Commissioner is hereby authorized to charge and credit Deposit Account No. 16-2372 as described below. A duplicate copy of this sheet is enclosed.
   ☐ Charge the amount of as filing fee.
   ☒ Credit any overpayment.
   ☐ Charge any additional filing fees required under 37 C.F.R. 1.16 and 1.17.
   ☐ Charge the issue set forth in 37 C.F.R. 1.18 at the mailing of the Notice of Allowance, pursuant to 37 C.F.R. 1.311(b).

Dated: January 12, 1999

Respectfully submitted,

Raymond H. J. Powell, Jr.
Registration No. 34,231
(703) 706-5862/3

PTO/SB/09 (12-97)
Approved for use through 9/30/00. OMB 0651-0031
Patent and Trademark Office; U S DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

STATEMENT CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) & 1.27(b))--INDEPENDENT INVENTOR	Docket Number (Optional) **RP990001US**

Applicant, Patentee, or Identifier: **Ford OXAL**

Application or Patent No.: **New Application**

Filed or Issued: **New Application**

Title: **METHOD FOR INTERACTIVELY VIEWING FULL-SURROUND IMAGE DATA AND APPARATUS THEREFOR**

As a below named inventor, I hereby state that I qualify as an independent inventor as defined in 37 CFR 1.9(c) for purposes of paying reduced fees to the Patent and Trademark Office described in:

- [X] the specification filed herewith with title as listed above.
- [ ] the application identified above.
- [ ] the patent identified above.

I have not assigned, granted, conveyed, or licensed, and am under no obligation under contract or law to assign, grant, convey, or license, any rights in the invention to any person who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person had made the invention, or to any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

Each person, concern, or organization to which I have assigned, granted, conveyed, or licensed or am under an obligation under contract or law to assign, grant, convey, or license any rights in the invention is listed below:

- [ ] No such person, concern, or organization exists.
- [ ] Each such person, concern, or organization is listed below.

Separate statements are required from each named person, concern, or organization having rights to the invention stating their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

Ford OXAAL		
NAME OF INVENTOR	NAME OF INVENTOR	NAME OF INVENTOR
Signature of inventor	Signature of inventor	Signature of inventor
Date	Date	Date

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

METHOD FOR INTERACTIVELY VIEWING FULL-SURROUND IMAGE DATA AND APPARATUS THEREFOR

BACKGROUND OF THE INVENTION

The present invention relates generally to a method and corresponding apparatus for viewing images. More specifically, the present invention relates to a method and corresponding apparatus for viewing full-surround, e.g., spherical, image data.

Systems and techniques for changing the perspective of a visible image in producing a resultant image, or systems and methods of transforming an image from one perspective form to another have been the subject of scientific thought and research for many years. Systems and techniques for transforming visible images can generally be divided into three separate categories:

(1) perspective generation systems and methods suitable for applications such as flight simulators;

(2) three-dimensional (3D) to two-dimensional (2D) conversion systems and methods; and

(3) miscellaneous systems and methods.

The first category includes U.S. Patent No. 3,725,563, which discloses a method of and apparatus for raster scan transformations using rectangular coordinates which are suitable for electronically generating images for flight simulators and the like. More specifically, the patent discloses a technique for raster shaping, whereby an image containing information from one viewpoint is transformed to a simulated image from another viewpoint. On the other hand, U.S. Patent No. 4,763,280 discloses a curvilinear dynamic image generation system for projecting rectangular coordinate images onto a spherical display surface. In the disclosed system, rectangular coordinates are converted to spherical coordinates and then the spherical coordinates are distorted for accomplishing the desired simulation of curvature.

The second category of systems and techniques perform 3D-to-2D conversion, or vice versa. For example, U.S. Patent No. 4,821,209 discloses a method of and apparatus for data transformation and clipping in a graphic display system, wherein data transformation is accomplished by matrix multiplication. On the other hand, U.S. Patent No. 4,667,236 discloses a television perspective effects system for providing perspective projection whereby each point of a three-dimensional object is projected onto a two-dimensional plane. New coordinates X' and Y' are prepared from the original coordinates X, Y and Z, and the viewing distance D, using the general formulas X'=XD/Z and Y'=YD/Z. As the object to be displayed is rotated around the X or Y axis, the viewing distance D is changed for each point.

In the third category, miscellaneous systems and methods are disclosed by, for example, U.S. Patent No. 5,027,287, which describes a device for the digital processing of images to obtain special geometrical effects wherein digital image data corresponding to intersection points on a rectangular X,Y grid are transposed by interpolation with respect to intersection points of a curved surface. U.S. Patent No. 4,882,679, on the other hand, discloses a system and associated method of reformatting images for three-dimensional display. The disclosed system is particularly useful for generating three-dimensional images from data generated by diagnostic equipment, such as magnetic resonance imaging.

However, none of the above described methods or systems permit viewing in circular perspective, which is the best way to view spherical data. It does all that linear perspective does when zoomed in, but it allows the view to zoom out to the point where the viewer can see almost everything in the spherical data simultaneously in a visually palatable and coherent way.

What is needed is a method for viewing full-surround, e.g., spherical, image data employing circular perspective. Moreover, what is needed is an apparatus for viewing full-surround, e.g., spherical, image data employing circular perspective. What is also needed is a method for viewing full-surround, e.g., spherical, image data employing circular perspective which is computationally simple. Preferably, the method for viewing full-surround, e.g., spherical, image data employing

circular perspective can be employed on any personal computer (PC) system possessing a three dimensional (3-D) graphics capability.

SUMMARY OF THE INVENTION

Based on the above and foregoing, it can be appreciated that there presently exists a need in the art for a viewing methods and corresponding apparatuses which overcome the above-described deficiencies. The present invention was motivated by a desire to overcome the drawbacks and shortcomings of the presently available technology, and thereby fulfill this need in the art.

The present invention implements a novel and practical circular perspective viewer for spherical data. Moreover, it implements the circular perspective viewer within the context of existing 3D graphics utilities native to personal computers (PCs). Thus, the method and corresponding apparatus for circular perspective viewing is practical for a broad market.

One object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data.

Another object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data onto a p-surface whereby the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from a point Q inside the region X of the p-surface.

Still another object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data wherein the viewer is allowed to interactively rotate the model.

Yet another object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data wherein the viewer is allowed to interactively change the direction of vision.

A still further object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data, wherein the viewer is allowed to interactively alter the focal length or view angle.

Another object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data, wherein the viewer is allowed to interactively move the viewpoint.

Still another object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data, wherein the viewpoint is close to the surface of the p-sphere.

Another object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data, wherein the viewer is allowed to interactively move the viewpoint.

A further object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data, wherein the viewer is allowed to select an area of the image and cause another model of the visible world to be loaded into said viewing system.

Another object according to the present invention is to provide a method and corresponding apparatus for modeling the visible world by texture mapping full-surround image data, wherein the viewer is allowed to perform any combination of actions specified immediately above.

It will be appreciated that none of the above-identified objects need actually be present in invention defined by the appended claims. In other words, only certain, and not all, objects of the invention have been specifically described above. Numerous other objects advantageously may be provided by the invention, as defined in the appended claims, without departing from the spirit and

scope of the invention.

These and other objects, features and advantages according to the present invention are provided by a method of modeling of the visible world using full-surround image data. Preferably, the method includes steps for selecting a view point within a p-surface, and texture mapping full-surround image data onto the p-surface such that the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from the view point to thereby generate a texture mapped p-surface.

According to one aspect of the invention, the method also includes a step for either rotating the texture mapped p-surface or changing the direction of view to thereby expose a new portion of the texture mapped p-surface. According to another aspect of the invention, a first the texture mapped p-sphere is replaced by a second texture mapped p-sphere by interactively selecting the new viewpoint from viewpoints within the second texture mapped p-sphere.

These and other objects, features and advantages according to the present invention are provided by a method of modeling of the visible world using full-surround image data, the method comprising steps for providing the full surround image data, selecting a view point within a p-surface, texture mapping full-surround image data onto the p-surface such that the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from the view point to thereby generate a texture mapped p-surface, and displaying a predetermined portion of the texture mapped p-sphere.

These and other objects, features and advantages according to the present invention are provided by an apparatus for modeling the visible world using full-surround image data, comprising first circuitry for selecting a view point within a p-surface, second circuitry for texture mapping full-surround image data onto the p-surface such that the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from the view point to thereby generate a texture mapped p-surface, and third circuitry for displaying a predetermined portion of

the texture mapped p-sphere.

These and other objects, features and advantages of the invention are disclosed in or will be apparent from the following description of preferred embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

These and various other features and aspects of the present invention will be readily understood with reference to the following detailed description taken in conjunction with the accompanying drawings, in which like or similar numbers are used throughout, and in which:

Fig. 1 illustrates a set of all rays from a predetermined viewpoint, which illustration facilitates an understanding of the present invention;

Fig. 2 illustrates a set of points, excluding the viewpoint, located on a corresponding one of the rays, which illustration facilitates an understanding of the present invention;

Fig. 3 illustrates the formation of a projection of the set of points, or a subset thereof, illustrated in Fig. 2;

Figs. 4A and 4B illustrate the resultant images generated by two different projections, respectively, given a constant viewpoint;

Fig. 5 illustrates the concept of linear perspective:

Fig. 6 illustrates the concept of circular perspective;

Fig. 7 illustrates the concept of stereographic projection;

Fig. 8 is a high level block diagram of a circular perspective viewing system according to the present invention;

Figs. 9A through 9G collectively form a listing of the dedicated code for converting a general purpose computer system into the circular perspective viewing system illustrated in Fig. 8; and

Figs. 10A and 10B collectively forming a listing of an exemplary code block for triangulating a hemisphere and texture coordinates.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The method and corresponding apparatus according to the present invention are similar to

that disclosed in U. S. Patent No. 5,684,937, which patent is incorporated herein by reference for all purposes, in that it generates perspective views derived for constants less than or equal to two, and greater than or equal to one, i.e., $1.0 \leq X \leq 2.0$. However, it will be appreciated that the inventive method and apparatus are different from U. S. Patent No. 5,684,937 in that the method for deriving perspective is different than 'explicitly' dividing all angles by a selected constant, as disclosed in that patent. Instead, the angles are 'implicitly' divided by a constant by moving the viewpoint around inside a "p-sphere". Additional details will be provided below.

By employing the method and corresponding apparatus according to the present invention, it is possible to create a virtual pictosphere using a conventional 3-D graphics system. Preferably, the inventive method an apparatus texture map the visible world onto a sphere. It should be mentioned that when the user selects a viewpoint at the center of this sphere and renders the view using the primitives of a conventional 3D graphics system, the used implicitly divides all angles by one, and result a linear perspective view. However, when the user selects a viewpoint on the surface of this sphere, selects a direction of view towards the center, and renders the view using the primitives of a conventional 3D graphics system, the user implicitly divides all angles by two, thus creating a circular perspective view. Moreover, by allowing the viewpoint to move around within or on the sphere, the user achieves results virtually identical to those achieved by U. S. Patent No. 5,684,937 for constants ranging from 1.0 to 2.0.

It will be appreciated that the method and corresponding apparatus according to the present invention implement a novel and practical circular perspective viewer of spherical data. The inventive method and apparatus advantageously can be achieved within the context of existing 3-D graphics utilities and hardware native to PCs. It will be noted from the statement immediately above that the inventive method and apparatus advantageously can be implemented in a broad range of existing systems.

The method and corresponding apparatus according to the present invention are predicated on the following starting, i.e., given, conditions:

(1) the set of all rays V from a given point VP, as illustrated in Fig. 1;

(2) a set of points P not including VP, each point in P being contained by one and only one ray in V, as illustrated in Fig. 2; and

(3) the set of color values C, each color in C being associated with one and only one ray in V, and also thereby associated with the point in P contained by said ray.

Moreover, the following definitions apply:

(1) POINTS P: The visible world.

(2) A PROJECTION OF P: A subset of points P. Any number of points Pn contained in P may be slid closer to or further from point VP along their corresponding rays. The resultant new configuration of points P is called a projection of P. The concept can best be understood by referring to Fig. 3;

(3) MAGIC POINT, VIEWPOINT, OR POINT OF PROJECTION: Point VP. Please note, no matter how points P are projected, their appearance will remain the same when viewed from point VP. This latter concept may best be understood by referring to Figs. 4A and 4B.

(4) FULL-SURROUND IMAGE DATA: data which samples the points P. This data encodes, explicitly or implicitly, the association of a color value with a given direction from a given point of projection. It should be mentioned that this point that full-surround image data is useful in many fields of entertainment because, when delivered to many viewers, it enables the construction of an independent viewing system defined below.

(5) P-SPHERE: a computer graphics representation of any polyhedron where there exists at least one point x inside (neither intersecting, nor lying outside) the polyhedron which may be connected to every point of the polyhedron with a distinct line segment, no portion of which said line segment lies outside the polyhedron or intersects the polyhedron at a point not an endpoint. The union of all such points x form the region X of the p-sphere. For a convex p-sphere, the region X is all points

of the interior of the p-sphere. Examples of computer graphics objects which may be modeled as p-spheres include a tetrahedron, a cube, a dodecahedron, and a faceted sphere.

(6) P-SURFACE: a computer graphics representation of any surface with a well-defined inside and outside, where there exists at least one point x inside (neither intersecting, nor lying outside) the surface which may be connected to every point of the surface with a distinct line segment, no portion of which said line segment lies outside the surface or intersects the surface at a point not an endpoint. The union of all such points x form the region X of the p-surface. For a convex p-surface, the region X is all points of the interior of the p-surface. Examples of computer graphics objects which may be modeled as p-surfaces: tetrahedron, cube, sphere, ellipsoid, cylinder, apple torus, lemon torus, b-spline surfaces closed or periodic in u and v. A p-sphere is a p-surface.

(7) LINEAR PERSPECTIVE: the projection of a portion of the visible world onto a plane or portion of a plane, as illustrated in Fig. 5.

(8) CIRCULAR PERSPECTIVE: the projection of the visible world, or portion thereof onto a plane, or a portion of a plane, after performing the perspective transformation of the visible world according to patent 5,684,937, where the constant used is 2. After such a transformation, when the direction of vision specified in said transformation is perpendicular to the projection plane, there is a one-to-one mapping of all the points P defining the visible world and all the points of an infinite plane. This definition is illustrated in Fig. 6;

(9) STEREOGRAPHIC PROJECTION: the one-to-one mapping of each point on a sphere to each point of an infinite tangent plane, said mapping performed by constructing the set of all rays from the antipodal point of the tangent point, the intersection of said rays with the plane defining said mapping. The understanding of this definition will be facilitated by reference to Fig. 7. Please note that circular perspective and stereographic projection produce geometrically similar (identically proportioned) mappings when the direction of vision specified in the perspective

transformation of circular perspective contains the tangent point specified in stereographic projection;

(10) INDEPENDENT VIEWING SYSTEM: an interactive viewing system in which multiple viewers can freely, independently of one another, and independently of the source of the image data, pan that image data in all directions with the effect that each viewer feels like they are "inside" of that imagery, or present at the location from which the imagery was produced, recorded, or transmitted; and

(11) STANDARD COMPUTER GRAPHICS SYSTEM: a computer graphics system which supports linear perspective viewing, including the changing of the focal length or the altering of the view angle, the apparent rotation of viewed objects, and/or the apparent changing of direction of vision, and the texture mapping of image data onto objects within the class of p-surface.

It will be appreciated that in a standard computer graphics system by texture mapping full-surround image data onto a p-surface such that the resultant texture map is effectively equivalent to projecting the full-surround imagery onto the p-surface from a some point Q contained in the region X of the p-surface, a representation of the visible world is achieved.

Referring to Figs. 9A through 9G, the method for viewing full-surround, e.g., spherical, image data will now be described. It should be mentioned that the corresponding code implementing the inventive method is written in the "C" language, although a plurality of programming languages are well known and readily adapted to this purpose. It will also be appreciated that code lines starting with "gl" or "glut" indicate calls to a conventional graphics library (GL) such as OpenGL™. One of ordinary skill in the art will readily appreciate that the last function in this listing is called main(). This is where the program starts.

It will be appreciated from inspection of Figs. 9A through 9G that the routine main() calls various glut... functions to set up GL, the graphics library used here. It will be noted that the glut... functions are part of GL. Furthermore, it will be appreciated that main() registers or maps certain

keyboard events with GL with glutKeyboardFunc(Key). In other words, glutKeyboardFunc(Key) defines the response of the inventive method to operation of the corresponding "Key."

Moreover, Key() is a function describing the actions of GL to keyboard events. Of importance are keyboard events ('z' and 'Z') which move the viewpoint in and out along the Z axis relative to the "p-sphere", effectively altering the perspective of the view, and keyboard events (55, 57, 52, 54, 56 and 50) which control rotation of the "p-sphere", effectively allowing the viewer to "look around".

It will be noted that main() registers the function display() with GL, with the glutDisplayFunc(display) function. Moreover, display() uses the global variables controlled by Key() to move the viewpoint along the Z axis relative to the "p-sphere", and to rotate the "p-sphere" relative to a constant direction of view.

Preferably, display() builds the "p-sphere" with glCallList(current_texture->tex1) and glCallList(current_texture->tex2). In the first instance, tex1 is mapped to a triangulation approximating a hemisphere, and is added to the display list. In the second instance, tex2 is mapped to the same hemisphere -- after rotating it 180 degrees to form a sphere -- and is also added to the display list, in the function readTexture(). Preferably, tex1 and tex2 are texture maps built from two pictures, respectively, taken with a fisheye lens. Advantageously, tex1 and tex2 collectively comprise a "pictosphere." It should be noted that the triangulation approximating the hemisphere was built in the function createHemisphere(), the full listing of which is found in Figs. 10A and 10B and not Figs. 9A - 9G.

At this point, Key() and display() have been registered with GL. The code main() then calls initialize_objects() which actually calls the routines readTexture() and createHemisphere(). All the other functions are support functions.

It will be appreciated that the user now has an instance of a p-sphere in GL made by mapping two fisheye images, e.g., photographs, to two adjoining hemispheres to thereby generate full-surround, e.g., spherical, image data. The user advantageously can interactively move the viewpoint away from the center of the p-sphere and, if so desired, very near the inside surface of the p-sphere. It should be mentioned at this point that the direction of view is still towards the center of the p-sphere. Moving the viewpoint from the center of the p-sphere automatically generates a circular perspective view, which advantageously can be displayed on display screen 20 of the PC illustrated in Fig. 8. Moving back to the center of the p-sphere, permits the user to generate a linear perspective view. It will also be appreciated from the discussion above that it is possible to rotate the surface of p-sphere, thus simulating looking around within the p-sphere.

It should be mentioned that by setting the viewpoint of the graphics system close to the center point of the p-sphere point and then enabling the viewer to rotate that p-sphere around a point close to the center point of the p-sphere, an independent viewing system providing linear perspective is achieved. Moreover, by adding the further capability of altering the focal length or angle of view, a zoom ability advantageously can be provided for the user.

It should also be mentioned that in the case where the p-surface used to model the visible world is a good approximation of a sphere, that is, a substantially better model than a tetrahedron or a cube, and where the view point of that representation is close to the approximate center of that p-surface, then by allowing the viewer to move the viewpoint away from center point to a point close to the surface of the p-surface, an independent viewing system is achieved in circular perspective. This is astounding when one considers that the native graphics system of a conventional PC only supports viewing in linear perspective. The method and correspond apparatus according to the present invention work because such an independent viewing system models stereographic projection, which is geometrically similar to circular perspective.

Furthermore, by letting the viewer move the viewpoint outside of the p-surface, the viewer

can get a feeling for how the independent viewing works. This can be useful for designers of systems containing many hyper-linked full-surround surfaces. For example, many p-spheres picturing the penthouse terraces of New York advantageously can be linked together so that the viewer may hop from p-sphere to p-sphere, simulating a tour of the terraces.

The above described method of the invention may be performed, for example, by the apparatus shown in FIG. 8. This viewing apparatus is composed of a central processing unit (CPU) 10 for controlling the components of the system in accordance with a control program stored in read-only memory (ROM) 60 or the like. The CPU 10 stores temporary data used during execution of the inventive method, i.e., viewing method, in random-access memory (RAM) 40. After the majority of the method steps are performed, the generated visible points are displayed on display device 20 (e.g., cathode ray tube (CRT) or liquid crystal display (LCD)) as visible points in accordance with the appropriate color values stored in color table 30, e.g., a color lookup table (CLUT) found in the graphics controller in most PCs. Advantageously, a spherical data generator device 70, such as a camera or the like, and preferably the data generator system disclosed in pending application serial No. 08/749,166 (filed November 14,1996; allowed May 6, 1998), which application is incorporated herein by reference for all purposes, may be used to generate different color values corresponding to the visible points of a viewed object, image or picture. An input device 50 is provided for entering data such as the viewer's viewing direction, reference plane, configuration (scaling) factor k, and other pertinent information used in the inventive viewing method.

As mentioned above, it will be appreciated that the method and corresponding apparatus according to the present invention advantageously can be used in an audiovisual or multimedia entertainment system. In particular, since each user advantageously can select his or her preferred view point and direction of view, multiple user can receive a single set of full-surround image data and generate corresponding multiple display images, in either linear or circular perspective.

Although presently preferred embodiments of the present invention have been described in

detail hereinabove, it should be clearly understood that many variations and/or modifications of the basic inventive concepts herein taught, which may appear to those skilled in the pertinent art, will still fall within the spirit and scope of the present invention, as defined in the appended claims.

```
#ifndef GGCONSTANTS_H
#define GGCONSTANTS_H


const double ggFourPi = 12.566371;  /* needs more digits */
const double ggTwoPi = 6.2831853;  /* needs more digits */
const double ggPi = 3.14159265358979323846;
const double ggHalfPi = 1.57079632679489661923;
const double ggThirdPi = 1.0471976; /* needs more digits */
const double ggQuarterPi = 0.78539816; /* needs more digits */
const double ggInversePi = 0.31830989;
const double ggSqrtTwo = 1.41421356237309504880;
const double ggInverseSqrtTwo = 0.70710678;
const double ggSqrtThree = 1.73205080756887729352;
const double ggSqrtFive = 2.23606797749978969640;
const double ggE = 2.71828182845904523536028747;

const double ggRad = 57.29577951308232;

#ifdef sun
const double ggInfinity = 1.0e10;
#else
#include <float.h>
const double ggInfinity = DBL_MAX;
#endif

#ifndef M_PI
#define M_PI ggPi
#endif

const double ggBigEpsilon = 0.0001;
const double ggEpsilon = 0.000001;
const double ggSmallEpsilon = 0.000000001;
const double ggTinyEpsilon = 0.000000000001;


const double ggColorRatio = 0.0039215686274509803;

#endif

#define ggMin(x,y) (x < y) ? x : y
#define ggMax(x,y) (x > y) ? x : y
```

APPENDIX PAGE 1

APPENDIX PAGE 2

```
/* Includes required */
#ifdef WINDOWS
#include <windows.h>
#endif

#include <GL\gl.h>
#include <GL\glut.h>
#include <stdio.h>
#include <math.h>
#ifdef USE_NETPBM
#include <ppm.h>
#endif

/**
 * something because of windows.
 */
void __eprintf() {
}

/**
 * our data structure of choice
 */
typedef struct obj {
    /* other parameters */
    float matrix[16];

    /* view angle */
    float viewangle;

    /* aspect ratio */
    float aspect;

    /* z of the camera */
    float tz;

    /* ry of the camera */
    float ry;
} Obj;

/* hold the display lists for textures */
typedef struct texture {
    int tex1;
    int tex2;
} Texture;

/**
 * our global variables
 */
/* camera settings */
Obj scene;

/* texture stuff */
Texture def;
Texture* current_texture = &def;

/* track the next display list number */
int nextDLnum = 2;

/* stuff for lighting */
float lightPos[4] = {2.0, 4.0, 2.0, 0};
float lightDir[4] = {0, 0, 1.0, 1.0};
float lightAmb[4] = {0.3, 0.3, 0.3, 1.0};
float lightDiff[4] = {0.6, 0.6, 0.6, 1.0};
float lightSpec[4] = {0.6, 0.6, 0.6, 1.0};
float clipDistance = 2.14;

int left, right, parent;
int width, height;
#ifdef USE_NETPBM
pixel** ppmPixels = 0;
#endif
GLubyte* sgiPixels = 0;
FILE* commands;
int doTakeSnapshot = 0;

#define HEMISPHERE 1
void createHemisphere(int listNum, int numPts, int geom);
void draw_left();
void draw_right();
void Key(unsigned char,int,int);

/**
 * read the frame buffer and write out a ppm file
 */
void takeSnapshot() {
#ifdef USE_NETPBM
    static int shotNum = 0;
    FILE* file;
    char name[50];
    int index, i, j;

    /* draw everything again */
    draw_right();
    glFlush();
    draw_left();
    glFlush();

    /* read the pixels from the frame buffer */
    glReadPixels(0, 0, width, height, GL_RGB, GL_UNSIGNED_BYTE, sgiPixels);

    /* convert them to the ppm */
    index = 0;
    for (i = height - 1; i >= 0; i--) {
        for (j = 0; j < width; j++) {
            PPM_ASSIGN(ppmPixels[i][j],
                       sgiPixels[index],
                       sgiPixels[index + 1],
                       sgiPixels[index + 2]);
            index += 3;
        }
    }

    /* open a file */
    sprintf(name, "%d.ppm", shotNum);
    shotNum++;
    file = fopen(name, "w");

    /* write the ppm file */
    ppm_writeppm(file, ppmPixels, width, height, 255, 0);

    /* close the file */
    fclose(file);
#endif
}

/**
 * Read in the ppm files and create display lists for a texture
 * returns the dimension of the image
 */
void readTexture(Texture* t, char* file1, char* file2) {
#ifdef USE_NETPBM
```

APPENDIX PAGE 3

```
FILE *fp1, *fp2;
int cols, rows, i, j, index;
pixel **map1, **map2;
GLubyte *tex1, *tex2;
pixval maxval;

/* open the files */
fp1 = fopen(file1, "r");
fp2 = fopen(file2, "r");
if (!fp1) {
    fprintf(stderr, "Couldn't open %s\n", file1);
}
if (!fp2) {
    fprintf(stderr, "Couldn't open %s\n", file2);
}

/* read the ppm files */
map1 = ppm_readppm(fp1, &cols, &rows, &maxval);
fprintf(stderr, "%s: rows = %d \t cols = %d\n", file1, rows, cols, maxval);
map2 = ppm_readppm(fp2, &cols, &rows, &maxval);
fprintf(stderr, "%s: rows = %d \t cols = %d\n", file2, rows, cols, maxval);

/* convert them */
tex1 = malloc(sizeof(GLubyte) * rows * cols * 3);
tex2 = malloc(sizeof(GLubyte) * rows * cols * 3);
index = 0;
for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
        /* R */
        tex1[index] = PPM_GETR(map1[i][j]);
        tex2[index] = PPM_GETR(map2[i][j]);
        index ++;

        /* G */
        tex1[index] = PPM_GETG(map1[i][j]);
        tex2[index] = PPM_GETG(map2[i][j]);
        index ++;

        /* B */
        tex1[index] = PPM_GETB(map1[i][j]);
        tex2[index] = PPM_GETB(map2[i][j]);
        index ++;
    }
}

/* create the textures in the left*/
glutSetWindow(left);
/* new display list*/
glNewList(nextDLnum, GL_COMPILE);
t->tex1 = nextDLnum;
nextDLnum++;
glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
             tex1);
glEndList();

/* new display list*/
glNewList(nextDLnum, GL_COMPILE);
t->tex2 = nextDLnum;
nextDLnum++;
glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
             tex2);
glEndList();

/* create the textures in the right*/
glutSetWindow(right);
    /* new display list*/
    glNewList(t->tex1, GL_COMPILE);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
                 tex1);
    glEndList();

    /* new display list*/
    glNewList(t->tex2, GL_COMPILE);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
                 tex2);
    glEndList();

    free(tex1);
    free(tex2);
    free(map1);
    free(map2);
#endif
}

/**
 * this will initialize the display lists for the objects
 */
void initialize_objects(int argc, char**argv) {
    float tmp[4];

    /* read in the texture */
    readTexture(&def, argv[1], argv[2]);

    /* create hemisphere left */
    glutSetWindow(left);
    createHemisphere(1, 50, GL_TRIANGLE_STRIP);

    /* create hemisphere right */
    glutSetWindow(right);
    createHemisphere(1, 50, GL_TRIANGLE_STRIP);

    /* scene */
    scene.viewangle = 50;
    scene.tz = 0;
}


void display_parent() {
    /* clear the screen */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}

void draw_left() {
    float tmp[4];

    /* clear the screen */
    glutSetWindow(left);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* adjust for scene orientation */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(scene.viewangle, scene.aspect, 0.1, 10.0);
    glRotatef(scene.ry, 1, 0, 0);
    glTranslatef(0, 0, scene.tz);

    /* draw our models */
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
```

APPENDIX PAGE 4

```
    /* now draw the semisphere */
    glEnable(GL_TEXTURE_2D);
    glColor3f(.5, .5, .5);
    glCallList(current_texture->tex1);
    glCallList(HEMISPHERE);

    glRotatef(180, 0.0, 0.0, 1.0);
    glColor3f(1, 1, 1);

    glCallList(current_texture->tex2);
    glCallList(HEMISPHERE);
    glPopMatrix();

    fprintf(stderr, "left - %s\n", gluErrorString(glGetError()));
}
void display_left()
{
    draw_left();
    glutSwapBuffers();
}

void draw_right() {
    float tmp[4];
    float height;

    /* clear the screen */
    glutSetWindow(right);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* setup the carera */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45, scene.aspect, clipDistance, 10.0);
    glTranslatef(0, 0, -3);
    glRotatef(15, 1, 0, 0);
    glRotatef(15, 0, 1, 0);
    glDisable(GL_TEXTURE_2D);

    /* draw our models */
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    /* draw a cube for the camera */
    glPushMatrix();
    glLoadIdentity();
    glRotatef(180, 1, 0, 0);
    glRotatef(-scene.ry, 1, 0, 0);
    glTranslatef(0, 0, scene.tz);

    tmp[0] = tmp[1] = tmp[2] = .8;
    tmp[3] = 1;
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    glutSolidCube(.1);

    /* now the light */
    tmp[0] = tmp[1] = tmp[2] = 0;
    tmp[3] = 1;
    glLightfv(GL_LIGHT1, GL_POSITION, tmp);
    glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, scene.viewangle / 2);
    tmp[0] = tmp[1] = 0; tmp[2] = 1; tmp[3] = 1;
    glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, tmp);

    /* draw a cone for the view frustrum */
    glLoadIdentity();
    height = 1 - scene.tz;
    glRotatef(-scene.ry, 1, 0, 0);
    glRotatef(45, 0, 0, 1);
    glTranslatef(0, 0, -1);
    tmp[0] = tmp[1] = .8;
    tmp[2] = 0;
    tmp[3] = 1.0;
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    glutWireCone(tan(scene.viewangle * 3.14 / 360.0) * height, height, 10, 4);
    glPopMatrix();

    /* draw one half of the sphere */
    tmp[0] = tmp[1] = tmp[2] = tmp[3] = 0;
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
    glEnable(GL_TEXTURE_2D);
    tmp[0] = tmp[1] = tmp[2] = tmp[3] = 1.0;
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    glCallList(current_texture->tex1);
    glCallList(HEMISPHERE);

    /* draw the other half */
    glRotatef(180, 0, 0, 1);
    glCallList(current_texture->tex2);
    glCallList(HEMISPHERE);
    glPopMatrix();
    fprintf(stderr, "right - %s\n", gluErrorString(glGetError()));
}

void display_right() {
    draw_right();
    glutSwapBuffers();
}

/*
 *  Handle Menus
 */
#define M_QUIT 1
void Select(int value)
{
    switch (value) {
    case M_QUIT:
        exit(0);
        break;
    }
    glutPostRedisplay();
}
void create_menu() {
    glutCreateMenu(Select);
    glutAddMenuEntry("Quit", M_QUIT);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}


/* Initializes hading model */
void init_left()
{
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);

    /* texture stuff */
    glEnable(GL_TEXTURE_2D);
```

```
    glPixelStorei(GL_UNPACK_ALIGNMENT, sizeof(GLubyte));
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
}

void init_right() {
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);

    /* texture stuff */
    glEnable(GL_TEXTURE_2D);
    glPixelStorei(GL_UNPACK_ALIGNMENT, sizeof(GLubyte));
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

    /* for blending */
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    /* lighting */
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiff);
    glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpec);

    glEnable(GL_LIGHT1);
    lightAmb[2] = 0;
    lightAmb[0] = lightAmb [1] = .8;
    lightAmb[3] = 1;
    glLightfv(GL_LIGHT1, GL_AMBIENT, lightAmb);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, lightDiff);
    glLightfv(GL_LIGHT1, GL_SPECULAR, lightSpec);
}

/*
 *  Called when the window is first opened and whenever
 *  the window is reconfigured (moved or resized).
 */
void myReshape(int w, int h)
{
    /* set width and height */
    width = w;
    height = h;

    /*  define the viewport */
    glViewport (0, 0, w, h);
    scene.aspect = 1.0*(GLfloat)w/2/(GLfloat)h;

    /* reshape the subwindows */
    /* first left */
    glutSetWindow(left);
    glutReshapeWindow(w/2, h);

    /* now right */
    glutSetWindow(right);
    glutReshapeWindow(w/2, h);
    glutPositionWindow(w/2, 0);
```

APPENDIX PAGE 5

```
    /* allocate memory for the snapshot thingy */
    if (sgiPixels) {
        free(sgiPixels);
    }
    sgiPixels = malloc(sizeof(GLubyte) * width * height * 3);

#ifdef USE_NETPBM
    if (ppmPixels) {
        ppm_freearray(ppmPixels, height);
    }
    ppmPixels = ppm_allocarray(width, height);
#endif
}

/**
 * an idle function to do automate things
 */
#define MOVE 0
#define SNAP 1
void idleSnapshots() {
    static moveOrSnap = MOVE;

    if (moveOrSnap == MOVE) {
        /* next command */
        Key('r', 0, 0);
        moveOrSnap = SNAP;
    } else {
        /* snapshot */
        Key(13, 0, 0);
        moveOrSnap = MOVE;
    }
}

/**
 * Rotate both displays
 */
void doRotate(float amt, float x, float y, float z) {
    glutSetWindow(left);
    glRotatef(amt, x, y, z);
    glutSetWindow(right);
    glRotatef(amt, x, y, z);
}

/*
 * Keyboard handler
 */
void
Key(unsigned char key, int x, int y)
{
    int keyInt;
    float matrix1[16];
    float matrix2[16];
    glutSetWindow(left);
    glMatrixMode(GL_MODELVIEW);
    glGetFloatv(GL_MODELVIEW_MATRIX, matrix1);
    glLoadIdentity();
    glutSetWindow(right);
    glMatrixMode(GL_MODELVIEW);
    glGetFloatv(GL_MODELVIEW_MATRIX, matrix2);
    glLoadIdentity();

    /* check for read command */
    if (key == 'r') {
        fscanf(commands, "%d", &keyInt);
```

```
    key = (char) keyInt;
    fprintf(stderr, "read %d - %c \n", key, key);
}

printf("%d\n", key);
fflush(stdout);
switch (key) {
case '!':
    /* register idle function */
    glutSetWindow(parent);
    glutIdleFunc(idleSnapshots);
    break;
case 13:
    /* press enter - take snapshot*/
    doTakeSnapshot = 1;
    break;
case 'c':
    clipDistance -= .02;
    break;
case 'C':
    clipDistance += .02;
    break;
case 'y':
    printf("ry = %f\n", scene.ry);
    scene.ry -= 5;
    break;
case 'Y':
    scene.ry += 5;
    break;
case 'z':
    scene.tz -= .02;
    break;
case 'Z':
    scene.tz += .02;
    break;
case 'a':
    scene.viewangle -= 1;
    break;
case 'A':
    scene.viewangle += 1;
    break;
case 55:
    doRotate(-5, 0.0, 0.0, 1.0);
    break;
case 57:
    doRotate(5, 0.0, 0.0, 1.0);
    break;
case 52:
    doRotate(-5, 0.0, 1.0, 0.0);
    break;
case 54:
    doRotate(5, 0.0, 1.0, 0.0);
    break;
case 56:
    doRotate(5, 1.0, 0.0, 0.0);
    break;
case 50:
    doRotate(-5, 1.0, 0.0, 0.0);
    break;
case '?':
    fprintf(stderr, "arrows - rotate the sphere\n");
    fprintf(stderr, "a/A viewangle\n");
    fprintf(stderr, "c/C adjust clip plane in right window\n");
    fprintf(stderr, "z/Z camera position\n");
    fprintf(stderr, "Escape quits \n");
```

APPENDIX PAGE (

```
    break;
case 27:            /* Esc will quit */
    exit(1);
    break;
default:
    fprintf(stderr, "Unbound key - %d  \n", key);
    break;
}
fprintf(stderr, "clip = %f  viewangle = %f  zdepth = %f \n",
        clipDistance, scene.viewangle, scene.tz);
glutSetWindow(left);
glMultMatrixf(matrix1);
glutPostRedisplay();
glutSetWindow(right);
glMultMatrixf(matrix2);
glutPostRedisplay();

/* check for snapshot */
if (doTakeSnapshot) {
    takeSnapshot();
    doTakeSnapshot = 0;
}
}


/*
 *  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGBA);

    /* create the parent window */
    parent = glutCreateWindow (argv[0]);
    width = 512; height = 256;
    glutReshapeWindow(width, height);
    glutKeyboardFunc(Key);
    glutReshapeFunc (myReshape);
    glutDisplayFunc(display_parent);
    create_menu();

    /* left window */
    left = glutCreateSubWindow (parent, 0, 0, width / 2, height);
    glutKeyboardFunc(Key);
    glutDisplayFunc(display_left);
    create_menu();
    init_left();

    /* right window */
    right = glutCreateSubWindow(parent, width / 2, 0, width / 2, height);
    glutKeyboardFunc(Key);
    glutDisplayFunc(display_right);
    create_menu();
    init_right();

    /* create objects */
    initialize_objects(argc, argv);

    /* open file */
    if (argc == 4) {
        fprintf(stderr, "Opening %s for commands\n", argv[3]);
        commands = fopen(argv[3], "r");
      }
      glutMainLoop();
  }
```

APPENDIX PAGE 7

```
/*****************************/
/* warp.c                    */
/*****************************/

#include <math.h>
#include "ggConstants.h"

/*
 * This function takes a point in the unit square, and maps it to
 * a point on the unit hemisphere.
 *
 * Copyright 1994 Kenneth Chiu
 *
 * This code may be freely distributed and used for any purpose, commercial
 * or non-commercial as long as attribution is maintained.
 */
void
map(double x, double y, double *x_ret, double *y_ret, double *z_ret) {

    double xx, yy, offset, theta, phi;

    x = 2*x - 1;
    y = 2*y - 1;

    if (y > -x) {               /* Above y = -x */
        if (y < x) {                /* Below y = x */
            xx = x;
            if (y > 0) {                /* Above x-axis */

/*              cerr<<"Octant 1 ";*/

                offset = 0;
                yy = y;
            } else {                    /* Below and including x-axis */
/*                  cerr<<"Octant 8 ";*/
                offset = (7*ggPi)/4;
                yy = x + y;
            }
        } else {                    /* Above and including y = x */
            xx = y;
            if (x > 0) {                /* Right of y-axis */

/*                  cerr<<"Octant 2 ";*/

                offset = ggPi/4;
                yy = (y - x);
            } else {                    /* Left of and including y-axis */

/*                  cerr<<"Octant 3 ";*/

                offset = (ggTwoPi)/4;
                yy = -x;
            }
        }
    } else {                    /* Below and including y = -x */
        if (y > x) {                /* Above y = x */
            xx = -x;
            if (y > 0) {                /* Above x-axis */

/*                  cerr<<"Octant 4 ";*/

                offset = (3*ggPi)/4;
                yy = -x - y;
            } else {                    /* Below and including x-axis */

/*                  cerr<<"Octant 5 ";*/

                offset = (ggFourPi)/4;
                yy = -y;
            }
        } else {                    /* Below and including y = x */
            xx = -y;
            if (x > 0) {                /* Right of y-axis */

/*                  cerr<<"Octant 7 ";*/

                offset = (6*ggPi)/4;
                yy = x;
            } else {                    /* Left of and including y-axis */
                if (y != 0) {

/*                      cerr<<"Octant 6 ";*/

                    offset = (5*ggPi)/4;
                    yy = x - y;
                } else {

/*                      cerr<<"Origin ";*/

                    *x_ret = 0;
                    *y_ret = 1;
                    *z_ret = 0;
                    return;
                }
            }
        }
    }

    theta = acos(1 - xx*xx);
    phi = offset + (ggPi/4)*(yy/xx);

    *x_ret = sin(theta)*cos(phi);
    *y_ret = cos(theta);
    *z_ret = sin(theta)*sin(phi);
}

/*
 * This function takes a point in the unit hemisphere, and maps it to
 * a point on the unit square.
 *
 * Copyright 1994 Keneth Chiu and Kurt Zimmerman
 *
 *
 * This code may be freely distributed and used for any purpose, commercial
 * or non-commercial as long as attribution is maintained.
 */
void
unmap(double x, double y, double z, double *x_ret, double *y_ret)
{
        double xx, yy, offset, theta, phi;

        theta = acos(y);

        if(theta < .0000001)   /* vertical center */
        {
                *x_ret = 0.5;
                *y_ret = 0.5;
                return;
        }
        else
```

```
{
        double cosphi,sinphi;
        cosphi = x/sin(theta);
        cosphi = ggMin(cosphi, 1.0); /* hack for now */
        cosphi = ggMax(cosphi, -1.0);
        sinphi = z/sin(theta);
        sinphi = ggMin(sinphi, 1.0); /* hack for now */
        sinphi = ggMax(sinphi, -1.0);

        if(sinphi >= 0 )
          phi = acos(cosphi);
        else if (sinphi < 0 && cosphi< 0)
          phi = -acos(cosphi);
        else
          phi = asin(sinphi);

        xx = sqrt(1 - y);

        if(phi < -(3 * M_PI)/4)
        {
                /* cerr<<"5th octant  ";*/
                yy = ((phi + M_PI) * xx)/(M_PI/4);
                *x_ret = -xx;
                *y_ret = -yy;
        }
        else if(phi <  -M_PI/2)
        {
                /*cerr<<"6th octant  ";*/
                yy = ((phi + (3*M_PI/4)) * xx)/(M_PI/4);
                *y_ret = -xx;
                *x_ret = *y_ret + yy;
        }
        else if(phi < -(M_PI)/4)
        {
                /*cerr<<"7th octant  ";*/
                yy = ((phi + M_PI/2) * xx)/(M_PI/4);
                *x_ret = yy;
                *y_ret = -xx;
        }
        else if(phi < 0)
        {
                /*cerr<<"8th octant  ";*/
                yy = ((phi + M_PI/4) * xx)/(M_PI/4);
                *x_ret = xx;
                *y_ret = yy - *x_ret;
        }
        else if(phi <  (M_PI)/4)
        {
                /*cerr<<"1st octant  ";*/
                yy = ((phi) * xx)/(M_PI/4);
                *x_ret = xx;
                *y_ret = yy;
        }
        else if(phi < M_PI/2)
        {
                /*cerr<<"2nd octant  ";*/
                yy = ((phi - M_PI/4) * xx)/(M_PI/4);
                *y_ret = xx;
                *x_ret = *y_ret - yy;
        }
        else if(phi < 3*M_PI/4)
        {
                /*cerr<<"3rd octant  ";*/
                yy = ((phi - M_PI/2) * xx)/(M_PI/4);
                *x_ret = -yy;
                *y_ret = xx;
        }
        else
        {
                /*cerr<<"4th octant  ";*/
                offset = 3*M_PI/4;
                yy = ((phi - offset) * xx)/(M_PI/4);
                *x_ret = -xx;
                *y_ret = -yy - *x_ret;
        }

}
*y_ret = (*y_ret + 1)/(2 + ggEpsilon);
*x_ret =(*x_ret + 1)/(2 + ggEpsilon);
}
```

APPENDIX PAGE 8

```
/*****************************/
/* warp.h                    */
/*****************************/

#ifndef WARP_DOTH
#define WARP_DOTH

void map(double x, double y, double *x_ret, double *y_ret, double *z_ret);
void unmap(double x, double y, double z, double *x_ret, double *y_ret);


#endif
```

APPENDIX PAGE 9

WHAT IS CLAIMED IS:

1. A method of modeling of the visible world using full-surround image data, said method comprising steps for:

selecting a view point within a p-surface; and

texture mapping full-surround image data onto said p-surface such that the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from said view point to thereby generate a texture mapped p-surface.

2. The method as recited in claim 1, further comprising the step of rotating said texture mapped p-surface so as to simulate rotating the direction of view in the opposite direction.

3. The method as recited in claim 1, wherein said selecting step comprises selecting the view point and a direction of view, and wherein said method further comprises the step of interactively changing said direction of view to thereby expose a corresponding portion of said texture mapped p-surface.

4. The method as recited in claim 1, further comprising the step of displaying a predetermined portion of said texture mapped p-surface.

5. The method as recited in claim 4, wherein the viewer is allowed to interactively alter at least one of focal length or an angle of view relative said textured mapped p-surface to thereby vary the displayed portion of said texture mapped p-surface.

6. The method as recited in claim 1, further comprising the steps of:

selecting a new viewpoint; and

repeating said texture mapping step using said new viewpoint.

7. The method as recited in claim 6, wherein the resultant viewpoint is close to the surface of said p-sphere.

8. The method as recited in claim 6, wherein said selecting step comprises interactively selectively said new viewpoint.

9. The method as recited in claim 8, wherein a first said texture mapped p-sphere is replaced by a second texture mapped p-sphere by interactively selecting said new viewpoint from viewpoints within said second texture mapped p-sphere.

10. A method of modeling of the visible world using full-surround image data, said method comprising steps for:

providing said full surround image data;

selecting a view point within a p-surface;

texture mapping full-surround image data onto said p-surface such that the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from said view point to thereby generate a texture mapped p-surface; and

displaying a predetermined portion of said texture mapped p-sphere.

11. The method as recited in claim 10, further comprising the step of rotating said texture mapped p-surface so as to simulate rotating the direction of view in the opposite direction.

12. The method as recited in claim 10, wherein said selecting step comprises selecting the view point and a direction of view, and wherein said method further comprises the step of interactively changing said direction of view to thereby display another portion of said texture mapped p-surface.

13. The method as recited in claim 10, further comprising the steps of:

selecting a new viewpoint; and

repeating said texture mapping step using said new viewpoint.

14. The method as recited in claim 13, wherein said selecting step comprises interactively selectively said new viewpoint.

15. The method as recited in claim 13, wherein a first said texture mapped p-sphere is replaced by a second texture mapped p-sphere by interactively selecting said new viewpoint from viewpoints within said second texture mapped p-sphere.

16. An apparatus for modeling the visible world using full-surround image data, comprising:

means for selecting a view point within a p-surface;

means for texture mapping full-surround image data onto said p-surface such that the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from said view point to thereby generate a texture mapped p-surface; and

means for displaying a predetermined portion of said texture mapped p-sphere.

17. The apparatus as recited in claim 16, wherein said selecting means comprises means for selecting said view point and interactively selecting a direction of view to thereby interactively display portions of said texture mapped p-surface.

18. The apparatus as recited in claim 16, wherein said selecting means provides for interactive selection of said viewpoint.

19. The apparatus as recited in claim 17, further comprising means for replacing a first said texture mapped p-sphere by a second texture mapped p-sphere by interactively selecting said viewpoint from a plurality of viewpoints within said second texture mapped p-sphere.

ABSTRACT OF THE DISCLOSURE

A method of modeling of the visible world using full-surround image data includes steps for selecting a view point within a p-surface, and texture mapping full-surround image data onto the p-surface such that the resultant texture map is substantially equivalent to projecting full-surround image data onto the p-surface from the view point to thereby generate a texture mapped p-surface. According to one aspect of the invention, the method also includes a step for either rotating the texture mapped p-surface or changing the direction of view to thereby expose a new portion of the texture mapped p-surface. According to another aspect of the invention, a first the texture mapped p-sphere is replaced by a second texture mapped p-sphere by interactively selecting the new viewpoint from viewpoints within the second texture mapped p-sphere. A corresponding apparatus is also described.

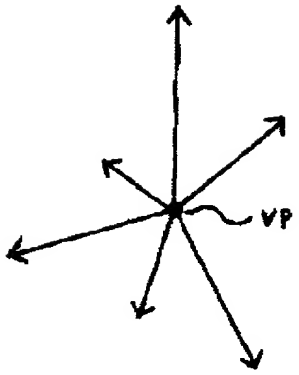


FIGURE 1.

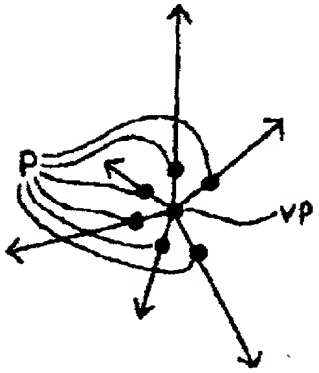


FIGURE 2.

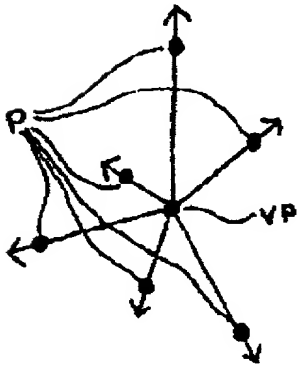


FIGURE 3.

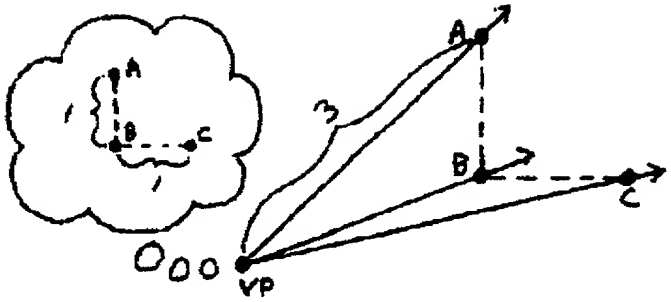


FIGURE 4A.

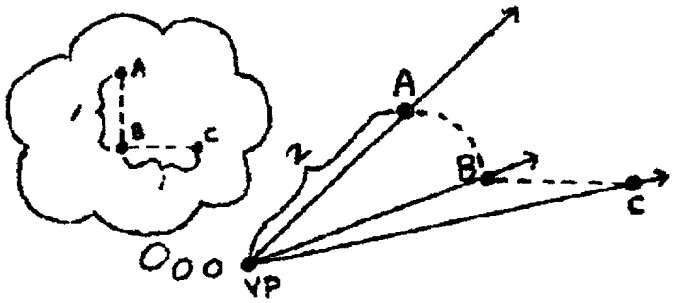


FIGURE 4B.

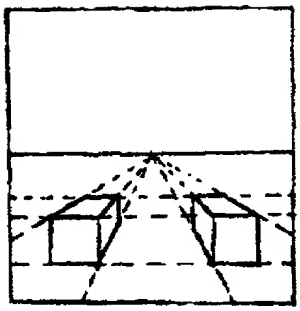


FIGURE 5.

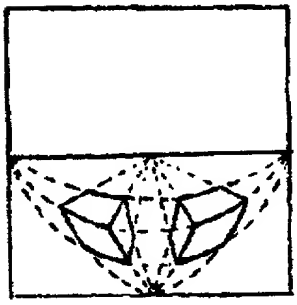


FIGURE 6.

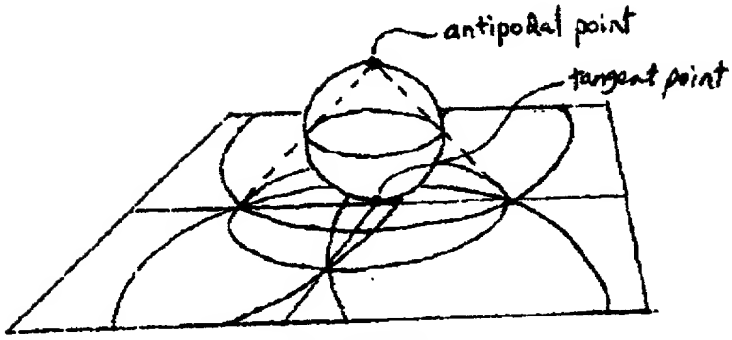
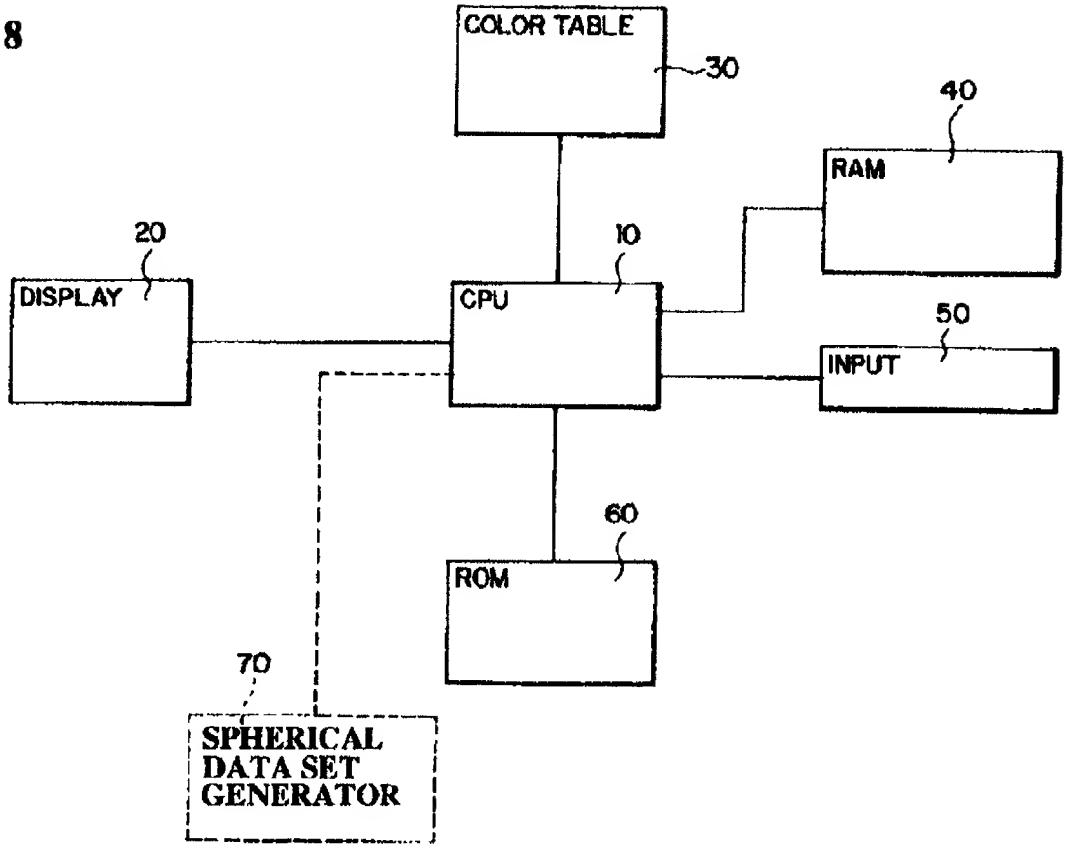


FIGURE 7.

Fig. 8



```
/* Includes required */
#include <GL/gl.h>
#include <GL/glut.h>
#include <stdio.h>
#include <ppm.h>
#include <math.h>

/**
 * something because of windows
 */
void __eprintf() {
}

/**
 * our data structure of choice
 */
typedef struct obj {
    /* other parameters */
    float matrix[16];

    /* view angle */
    float viewangle;

    /* aspect ratio */
    float aspect;

    /* z of the camera */
    float tz;

    /* ry of the camera */
    float ry;
} Obj;

/* hold the display lists for textures */
typedef struct texture {
    int tex1;
    int tex2;
} Texture;

/**
 * our global variables
 */
/* camera settings */
Obj scene;

/* texture stuff */
Texture def;
Texture* current_texture = &def;

/* track the next display list number */
int nextDLnum = 2;

/* stuff for lighting */
float lightPos[4] = {2.0, 4.0, 2.0, 0};
float lightDir[4] = {0, 0, 1.0, 1.0};
float lightAmb[4] = {0.4, 0.4, 0.4, 1.0};
float lightDiff[4] = {0.8, 0.8, 0.8, 1.0};
float lightSpec[4] = {0.8, 0.8, 0.8, 1.0};
int lights = 0;
int outsideView = 0;
int parent;


#define HEMISPHERE 1
void createHemisphere(int listNum, int numPts, int geom);
```

Fig. 9A

```
/**
 * Read in the ppm files and create display lists for a texture
 * returns the dimension of the image
 */
pixel **map1, **map2;
GLubyte *tex1, *tex2, **tmpPP, *tmpP;
void readTexture(Texture* t, char* file1, char* file2) {
    FILE *fp1, *fp2;
    int cols, rows, i, j, index;
    pixval maxval;

    /* open the files */
    fp1 = fopen(file1, "r");
    fp2 = fopen(file2, "r");
    if (!fp1) {
        fprintf(stderr, "Couldn't open %s\n", file1);
    }
    if (!fp2) {
        fprintf(stderr, "Couldn't open %s\n", file2);
    }

    /* read the ppm files */
    map1 = ppm_readppm(fp1, &cols, &rows, &maxval);
    fprintf(stderr, "%s: rows = %d \t cols = %d\n", file1, rows, cols, maxval);
    map2 = ppm_readppm(fp2, &cols, &rows, &maxval);
    fprintf(stderr, "%s: rows = %d \t cols = %d\n", file2, rows, cols, maxval);

    /* convert them */
    tex1 = malloc(sizeof(GLubyte) * rows * cols * 3);
    tex2 = malloc(sizeof(GLubyte) * rows * cols * 3);
    index = 0;
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            /* R */
            tex1[index] = PPM_GETR(map1[i][j]);
            tex2[index] = PPM_GETR(map2[i][j]);
            index ++;

            /* G */
            tex1[index] = PPM_GETG(map1[i][j]);
            tex2[index] = PPM_GETG(map2[i][j]);
            index ++;

            /* B */
            tex1[index] = PPM_GETB(map1[i][j]);
            tex2[index] = PPM_GETB(map2[i][j]);
            index ++;
        }
    }

    /* create the textures */
    /* new display list*/
    glNewList(nextDLnum, GL_COMPILE);
    t->tex1 = nextDLnum;
    nextDLnum++;
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
                 tex1);
    glEndList();

    /* new display list*/
    glNewList(nextDLnum, GL_COMPILE);
    t->tex2 = nextDLnum;
    nextDLnum++;
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
```

Fig. 9B

```
                tex2);
    glEndList();
}

/**
 * this will initialize the display lists for the objects
 */
void initialize_objects(int argc, char**argv) {
    float tmp[4];

    /* read in the texture */
    readTexture(&def, argv[1], argv[2]);

    /* create hemisphere */
    createHemisphere(1, 50, GL_TRIANGLE_STRIP);

    /* scene */
    scene.viewangle = 130;
    scene.tz = 0;
    scene.ry = 0;
}


/*
 *  Clear the screen. draw the objects
 */
void display()
{
    float tmp[4];
    float height;

    /* clear the screen */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* adjust for scene orientation */
    glMatrixMode(GL_PROJECTION);
    if (outsideView) {
        glLoadIdentity();
        gluPerspective(45, scene.aspect, 0.1, 10.0);
        glTranslatef(0, 0, -3);
        glRotatef(45, 1, 0, 0);
        glRotatef(45, 0, 1, 0);
        glDisable(GL_TEXTURE_2D);
        glColor3f(.8, .8, .8);
    } else {
        glLoadIdentity();
        gluPerspective(scene.viewangle, scene.aspect, 0.1, 10.0);
        glTranslatef(0, 0, scene.tz);
        glRotatef(scene.ry, 0, 1, 0);
    }

    /* draw our models */
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    if (outsideView) {
        /* transform to where the camera would be */
        glPushMatrix();

        /* draw a cube for the camera */
        glLoadIdentity();
        glRotatef(180, 1, 0, 0);
        glTranslatef(0, 0, scene.tz);
        tmp[0] = tmp[1] = tmp[2] = .8;
        tmp[3] = 1;
```

Fig. 9C

```
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
        glutSolidCube(.1);

        /* draw a cone for the view frustrum */
        glLoadIdentity();
        height = 1 - scene.tz;
        glRotatef(45, 0, 0, 1);
        glTranslatef(0, 0, -1);
        tmp[0] = tmp[1] = 1;
        tmp[2] = 0;
        tmp[3] = .3;
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
        glutSolidCone(tan(scene.viewangle * 3.14 / 360.0) * height, height, 20, 1);
        glPopMatrix();
        glEnable(GL_TEXTURE_2D);
    }

    /* now draw the semisphere */
    if (lights) {
        tmp[0] = tmp[1] = tmp[2] = .8;
        tmp[3] = .8;
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 10.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    }

    glCallList(current_texture->tex1);
    glCallList(HEMISPHERE);

    if (lights) {
        tmp[0] = tmp[1] = tmp[2] = .5;
        tmp[3] = .5;
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 10.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    }

    glRotatef(180.0, 0.0, 0.0, 1.0);
    glCallList(current_texture->tex2);
    glCallList(HEMISPHERE);
    glPopMatrix();

    fprintf(stderr, "%s\n", gluErrorString(glGetError()));
    glutSwapBuffers();
}

/*
 *  Handle Menus
 */
#define M_QUIT 1
void Select(int value)
{
    switch (value) {
    case M_QUIT:
        exit(0);
        break;
    }
    glutPostRedisplay();
}
void create_menu() {
    fprintf(stderr, "Press ? for help\n");
```

Fig. 9D

```
    glutCreateMenu(Select);
    glutAddMenuEntry("Quit", M_QUIT);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}


/* Initializes hading model */
void myInit(void)
{
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);

    /* texture stuff */
    glPixelStorei(GL_UNPACK_ALIGNMENT, sizeof(GLubyte));
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glEnable(GL_TEXTURE_2D);

}

/*
 *  Called when the window is first opened and whenever
 *  the window is reconfigured (moved or resized).
 */
void myReshape(int w, int h)
{
    glViewport (0, 0, w, h);            /*  define the viewport */
    scene.aspect = 1.0*(GLfloat)w/(GLfloat)h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(scene.viewangle, scene.aspect, 0.1, 10.0);
    glMultMatrixf(scene.matrix);
    glMatrixMode (GL_MODELVIEW);        /* back to modelview matrix  */
}

/*
 * Keyboard handler
 */
void
Key(unsigned char key, int x, int y)
{
    float matrix[16];
    glMatrixMode(GL_MODELVIEW);
    glGetFloatv(GL_MODELVIEW_MATRIX, matrix);
    glLoadIdentity();
    fprintf(stderr, "%d - %c  ", key, key);
    switch (key) {
    case 'o':
        if (!outsideView {
            fprintf(stderr, "outside on  ");
            outsideView = 1;

            /* turn on blending */
            glEnable(GL_BLEND);
            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

            /* We want to see color */
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

            /* turn on our spotlight */
            glEnable(GL_LIGHT1);
            glLightfv(GL_LIGHT1, GL_AMBIENT, lightAmb);
```

Fig. 9E

```
            glLightfv(GL_LIGHT1, GL_DIFFUSE, lightDiff);
            glLightfv(GL_LIGHT1, GL_SPECULAR, lightSpec);
            glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, lightDir);
        } else {
            fprintf(stderr, "outside off  ");
            outsideView = 0;
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
            glDisable(GL_BLEND);
        }
        break;
    case 'F':
        fprintf(stderr, "flat  ");
        glShadeModel(GL_FLAT);
        break;
    case 'f':
        fprintf(stderr, "smooth  ");
        glShadeModel(GL_SMOOTH);
        break;
    case 'y':
        printf("ry = %f\n", scene.ry);
        scene.ry -= 5;
        break;
    case 'Y':
        scene.ry += 5;
        break;
    case 'z':
        scene.tz -= .02;
        fprintf(stderr, " tz = %f  ", scene.tz);
        break;
    case 'Z':
        scene.tz += .02;
        fprintf(stderr, " tz = %f  ", scene.tz);
        break;
    case 'a':
        scene.viewangle -= 1;
        fprintf(stderr, " angle: %f  ", scene.viewangle);
        break;
    case 'A':
        scene.viewangle += 1;
        fprintf(stderr, "angle: %f  ", scene.viewangle);
        break;
    case 55:
        glRotatef(-5, 0.0, 0.0, 1.0);
        break;
    case 57:
        glRotatef(5, 0.0, 0.0, 1.0);
        break;
    case 52:
        glRotatef(-5, 0.0, 1.0, 0.0);
        break;
    case 54:
        glRotatef(5, 0.0, 1.0, 0.0);
        break;
    case 56:
        glRotatef(5, 1.0, 0.0, 0.0);
        break;
    case 50:
        glRotatef(-5, 1.0, 0.0, 0.0);
        break;
    case 'q':
        if (lights) {
            glDisable(GL_LIGHT0);
            glDisable(GL_LIGHTING);
            lights = 0;
            fprintf(stderr, "no lights  ");
```

Fig. 9F

```
        } else {
            glEnable(GL_LIGHTING);
            glEnable(GL_LIGHT0);
            glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
            glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
            glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiff);
            glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpec);
            lights = 1;
            fprintf(stderr, "lights  ");
        }
        break;
    case 't':
        fprintf(stderr, "texture off  ");
        glDisable(GL_TEXTURE_2D);
        break;
    case 'T':
        fprintf(stderr, "texture on  ");
        glEnable(GL_TEXTURE_2D);
        break;
    case '?':
        fprintf(stderr, "hjkl - rotate current object\n");
        fprintf(stderr, "s/S - shrink / grow the object or zoom the scene\n");
        fprintf(stderr, "a/A viewangle\n");
        fprintf(stderr, "z/Z camera position\n");
        fprintf(stderr, "f/F flat smooth\n");
        fprintf(stderr, "Escape quits \n");
        break;
    case 27:            /* Esc will quit */
        exit(1);
        break;
    default:
        fprintf(stderr, "Unbound key - %d  ", key);
        break;
    }
    fprintf(stderr, "\n");
    glMultMatrixf(matrix);
    glutPostRedisplay();
}

/*
 *  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGBA);
    parent = glutCreateWindow (argv[0]);
    myInit();
    glutKeyboardFunc(Key);
    glutReshapeFunc (myReshape);
    glutDisplayFunc(display);
    create_menu();
    initialize_objects(argc, argv);
    glutMainLoop();
}
```

Fig. 9G

```
#ifdef WINDOWS
#include <windows.h>
#endif
#include <GL/gl.h>
#include <GL/glut.h>

#include "warp.h"
#include <stdio.h>
/**
 * Triangulate a hemisphere and texture coordinates.
 * listNum - display list number
 * numPts -  number of points to a side
 * return the display list
 */
void createHemisphere(int listNum, int numPts, int geom) {
    double incr = 1.0 / numPts;
    double u, v, x, y, z;
    float tx, tz;
    int i, j;

    /* start the display list */
    glNewList(listNum, GL_COMPILE_AND_EXECUTE);

    /* create the coordinates */
    /* use the square to circle map */
    /* across then down */
    v = 0;
    for (j = 0; j < numPts ; j++) {
        /* start the tri strip */
        glBegin(geom);
        u = 0;
        for (i = 0; i <= numPts; i++) {
            /* do the top point */
            /* get the XYZ coords */
            map(u, v, &x, &y, &z);

            /* create the texture coord */
            tx = x / 2 + .5;
            tz = z / 2 + .5;
            if (tx > 1.0 || tz > 1.0 || tx < 0.0 || tz < 0.0) {
                printf("not in range %f %f\n", tx, tz);
            }
            glTexCoord2f(tx, tz);

            /* normal */
            glNormal3f(x, y, z);

            /* create the coord */
            glVertex3f(x, y, z);

            /* get the XYZ coords */
            map(u, v + incr, &x, &y, &z);

            /* create the texture coord */
            tx = x / 2 + .5;
            tz = z / 2 + .5;
            if (tx > 1.0 || tz > 1.0 || tx < 0.0 || tz < 0.0) {
                printf("not in range %f %f\n", tx, tz);
            }
            glTexCoord2f(tx, tz);

            /* normal */
            glNormal3f(x, y, z);

            /* create the coord */
```

Fig. 10A

```
            glVertex3f(x, y, z);

            /* adjust u */
            u += incr;
        }
        /* done with the list */
        glEnd();

        /* adjust v */
        v += incr;
    }

    /* all done with the list */
    glEndList();
}
```

Fig. 10B

Please type a plus sign (+) inside this box → [+]

PTO/SB/01 (12-97)
Approved for use through 9/30/00. OMB 0651-0032
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION FOR UTILITY OR DESIGN PATENT APPLICATION (37 CFR 1.63)

[X] Declaration Submitted with Initial Filing OR [ ] Declaration Submitted after Initial Filing (surcharge (37 CFR 1.16 (e)) required)

Attorney Docket Number	**RP990001US**
First Named Inventor	**Ford OXAAL**
COMPLETE IF KNOWN	
Application Number	/
Filing Date	
Group Art Unit	
Examiner Name	

As a below named inventor, I hereby declare that:

My residence, post office address, and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

METHOD FOR INTERACTIVELY VIEWING FULL-SURROUND IMAGE DATA AND APPARATUS THEREFOR

(Title of the Invention)

the specification of which
[X] is attached hereto
OR
[ ] was filed on (MM/DD/YYYY) _____ as United States Application Number or PCT International Application Number _____ and was amended on (MM/DD/YYYY) _____ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate, or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Certified Copy Attached? YES	NO
			[ ]	[ ]	[ ]
			[ ]	[ ]	[ ]
			[ ]	[ ]	[ ]
			[ ]	[ ]	[ ]

[ ] Additional foreign application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto:

I hereby claim the benefit under 35 U.S.C. 119(e) of any United States provisional application(s) listed below.

Application Number(s)	Filing Date (MM/DD/YYYY)	
60/071,148	01/12/98	[ ] Additional provisional application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto.

Burden Hour Statement: This form is estimated to take 0.4 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

Please type a plus sign (+) inside this box → [+]

PTO/SB/01 (12-97)
Approved for use through 9/30/00. OMB 0651-0032
Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION — Utility or Design Patent Application

I hereby claim the benefit under 35 U.S.C. 120 of any United States application(s), or 365(c) of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. 112, I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application.

U.S. Parent Application or PCT Parent Number	Parent Filing Date (MM/DD/YYYY)	Parent Patent Number *(if applicable)*

☐ Additional U.S. or PCT international application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto

As a named inventor, I hereby appoint the following registered practitioner(s) to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith: ☐ Customer Number [ ] → *Place Customer Number Bar Code Label here*

OR

☒ Registered practitioner(s) name/registration number listed below

Name	Registration Number	Name	Registration Number
Robert A, Westerlund	31,439		
Raymond H. J. Powell, Jr.	34,231		
Ramon R. Hoch	34,108		

☐ Additional registered practitioner(s) named on supplemental Registered Practitioner Information sheet PTO/SB/02C attached hereto.

Direct all correspondence to: ☐ Customer Number or Bar Code Label [ ] *OR* ☒ Correspondence address below

Name	Westerlund & Powell, P.C.						
Address	122 N. Alfred Street						
Address							
City	Alexandria			State	VA	ZIP	22314-3011
Country	USA	Telephone	(703) 706-5862			Fax	(703) 706-5860

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Name of Sole or First Inventor: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any])	Family Name or Surname
Ford	OXAAL

Inventor's Signature								Date	
Residence: City	Cohoes		State	NY		Country	USA	Citizenship	US
Post Office Address	42 Western Avenue								
Post Office Address									
City	Cohoes	State	NY		ZIP	12047		Country	USA

☐ Additional inventors are being named on the ____ supplemental Additional Inventor(s) sheet(s) PTO/SB/02A attached hereto